Greedy Cache Management Techniques for Mobile Devices

Shahram Ghandeharizadeh and Shahin Shayandeh
Computer Science Department
University of Southern California
Los Angeles, CA 90089
shahram@usc.edu, shayande@usc.edu

Abstract

Mobile devices are configured with one or more wireless cards that provide limited radio-range and unreliable transmission. These energy-constrained devices are configured with a fixed amount of storage. A device may set aside a fraction of its local storage as a cache to minimize use of the network when servicing requests, enhancing metrics such as startup latency and data availability. In this paper, we focus on a repository of continuous media (audio and video) clips and study several greedy cache management techniques and their cache hit rates. This metric reflects what percentage of requests for clips is serviced when a mobile device is disconnected from the network. The device becomes network detached due to factors such as residing in a densely populated geographical location with over committed network bandwidth or travels to geographical areas with no base station coverage. We investigate repositories of both equi-sized and variable-sized clips, identifying limitations of the current techniques. Our primary contribution is development of three novel techniques to address these limitations. They are adaptable and provide competitive cache hit rates. One technique, called Dynamic Simple, provides a higher cache hit rate and adapts faster to changing patterns of access to clips when compared with other techniques.

1 Introduction

Advances in wireless communication, mass storage, and processing have enabled mobile devices that offer their users a variety of information services. Applications of these devices are diverse, ranging from entertainment to education and military. As an example, today's mobile telephone is a ubiquitous device that might act as both a camera and a camcorder, plays music using its speakers or earphones, displays video clips using its color screen, stores personal information and interface with a projector (or a computer screen) to deliver multimedia presentations, includes a browser to enable its user to roam the Internet and display content (e.g., YouTube video clips), and provides traditional voice services. The computer and communications industry is exploring "fixed-mobile convergence" (FMC) [8] where the mobile telephone replaces existing fixed-line telephone by detecting wireless Internet base stations and "roaming" onto it by switching to voice over IP (VoIP). This is to provide the user with reliability, low cost, and clear audio qualities comparable to the fixed-line telephone. In addition, FMC provides the convenience of a single handset, a single address book and a single voicemail box.

The FMC phone is expected to be configured with two types of network connections [8]: a Wi-Fi interface such as 802.11b or g, and traditional cellular. When on the road, its cellular connectivity employs the existing cellular base stations to provide voice and data services at bandwidths ranging from tens of Kilobits per second (Kbps) to a few Megabits per second (Mbps). When at home, the Wi-Fi connectivity detects the home base station that provides broadband network services, and employs it to route a call using VoIP. The home base station is a fixed-line broadband connection (such as DSL or cable modem), communicating with nearby mobile phones using Wi-Fi. This Wi-Fi connectivity provides bandwidths ranging from hundreds of Kbps to tens of Mbps. Both cellular and Wi-Fi connectivity are required because the radio range of the Wi-Fi connection is limited to tens of feet[1] while the cellular connection is in the order of miles.

Similar to today's camcorders, an FMC phone might be configured with an inexpensive magnetic disk drive, offering its users hundreds of Gigabytes of storage. With prerecorded data, some of this storage might be configured as a cache to enhance the availability of data and improve qual-

[1] WiMax, 802.16 [5], provides a longer radio-range with bandwidths comparable to 802.11b. While deployed in Japan, many are awaiting its arrival in the United States. WiMax may minimize the role of an in home broadband connection. We do not elaborate on this alternative because it is not central to the caching techniques explored in this paper.

ity of service metrics such as startup latency. Consider each in turn. Data availability is enhanced because the user may retrieve the cached data when the device is out of the range of both the cellular and Wi-Fi base stations. Startup latency is the time elapsed from when a user references a clip to the onset of its display. When streaming a clip from a base station, this metric includes network delays attributed to time spent to reserve bandwidth by negotiating with the base station. Bandwidth reservation and admission control are required for streaming media to ensure the mobile device does not starve for data. Otherwise, its display may suffer from frequent disruptions and delays, termed hiccups. Network delays are eliminated when a device observes a cache hit and services its requests using its local cache.

In this paper we study alternative caching techniques in order to maximize cache hit rate of a device. Caching techniques can be taxonomized along two dimensions: 1) those that manage either equi or variable sized data items, and 2) either greedy[2] or cooperative caching techniques. Consider each dimension in turn. Caching techniques based on fix sized objects or pages [7] are employed by operating systems and database management systems. Proxy cache servers must manage objects with different sizes [16]. As demonstrated in Section 3.3, a technique such as LRU that provides competitive cache hit rates with equi-sized objects may provide a low cache hit rate for variable sized objects. One contribution of this study is to identify techniques that manage both equi and variable sized objects effectively[3].

A greedy caching technique strives to optimize a local metric such as cache hit rate. It is in sharp contrast to a cooperative caching technique that strives to optimize a global metric by coordinating the caches of different mobile devices in the radio range of one another. A global metric might be the number of requests serviced using the cache of different mobile devices (instead of using the base station).

Cooperative caching was shown to improve the performance of file and virtual memory systems in a high-speed, local-area network environment [9]. In the context of proxy caches, it was shown: a) to improve performance among collections of small organizations, and b) unlikely to have significant benefits for larger organizations or populations [16].

In this study, we investigate greedy caching techniques for mobile devices such as FMC. An investigation of cooperative caching techniques is a future research direction. This is because an understanding of greedy techniques is required in order to develop and evaluate effective cooperative caching technique. Moreover, the performance of a greedy technique is key to evaluating a cooperative technique to determine if its complexity is justified.

One may evaluate different greedy caching techniques using the following metrics:

- Cache hit rate: the percentage of clip requests satisfied using the cache. A 90% hit rate means that 9 of every 10 clips referenced by the client are found in the cache.

- Cache byte hit rate: the number of bytes satisfied from the cache as a fraction of the total bytes referenced by the client. This metric represents the amount of work imposed on the network.

- Processor and network utilization quantify the amount of resources used by a caching technique. Processor utilization quantifies the complexity of a design and its implementation. Network utilization defines how much bandwidth is used by a technique and is a function of byte hit rate. Both impact the amount of power consumed by a device.

- Average startup latency: the delay incurred from when a client requests a clip to the onset of its display. Streaming a referenced clip from the cache minimizes startup latency because bandwidth of local storage is higher than the bandwidth required to display a clip ($B_{Display}$). When streaming a clip using the network, startup latency is dictated by the allocated network bandwidth ($B_{Delivery}$) and the overhead of admission control. Assuming a request is admitted and $B_{Delivery}$ is higher than $B_{Display}$, a client may start the display as soon as sufficient data is prefetched in the buffer to compensate for fluctuations in $B_{Delivery}$. If $B_{Delivery}$ is lower than $B_{Display}$ then the client device must prefetch enough data in order to compensate for the lower delivery rate. The amount of prefetch data is defined as [10]: $S_i - \lfloor \frac{B_{Delivery}}{B_{Display}} \times S_i \rfloor$.

- Throughput of a geographical region: the number of devices in a geographical area able to display their referenced clips simultaneously. If each device observes a cache hit then the throughput of the region equals the number of devices in that area. When devices in the same radio range do not find their referenced clips in their cache, they compete for the wireless network bandwidth. These requests are rejected once the network bandwidth is exhausted, reducing the throughput of that region.

[2] Greedy caching techniques might be categorized as recently-based, frequency-based, size-based, function-based, and randomized [17, 15].

[3] A naive technique to manage variable sized objects is as follows. Partition both the cache and each object into equi-sized blocks and use LRU-K to manage the cached blocks. The choice of block size is important with such a technique. If a block is larger than an object then this technique will waste cache space, reducing cache hit rate. A smaller block size reduces the impact of wasted space. However, it increases the amount of required book keeping information such as the time stamp of last block reference. Moreover, when compared with other techniques presented in this paper, it does not provide a higher cache hit rate than DYNSimple, see discussions of Figure 5.a.

While all metrics are important, in this study we focus on cache hit rate because it enhances availability of data when a device is in an area with either no base station coverage or so densely populated that the available bandwidth to the base station is exhausted. Thus, we do not consider algorithms that enhance other metrics by sacrificing cache hit rate. An example is GDS-Popularity [13] which enhances byte hit rate at the expense of cache hit rate.

The rest of this paper is organized as follows. Section 2 presents the problem statement. Subsequently, Section 3 provides an overview of one current off-line technique called Simple [11] and two on-line techniques, LRU-K [14] and GreedyDual [18, 3]. An off-line technique is provided with advance knowledge of future requests while an on-line technique is not previewed to this information. A comparison of these techniques in Section 3.3 shows the following tradeoffs: 1) While Simple provides the highest cache hit rate, it is not practical for use in a real system because it is an off-line technique, 2) LRU-K is ideal for managing equi-sized clips and provides low cache hit rates when clip sizes are different, 3) GreedyDual provides a low cache hit rate when clips are approximately the same size. These tradeoffs motivate us to develop variants of Simple, LRU-K, and GreedyDual in Section 4. These variants resolve the limitations of their original counter-parts. We show all three variants adjust to dynamic changes in frequency of access. When focusing on cache hit rate, the variant based on Simple, called DYNSimple, outperforms GreedyDual consistently. Moreover, DYNSimple adjusts to fluctuating access patterns quicker than the other alternative. Brief conclusions and future research directions are offered in Section 5

2 Problem statement

The caching problem can be stated as follows. Given a sequence of clip references, where each reference has a time stamp and each clip has a size, maintain a cache of clips on a client device with the objective to maximize number of client requests that find their referenced clips in cache. The cache has a fixed size S_T. The size of clip repository is S_{DB} and larger than the cache size, $S_{DB} > S_T$. Otherwise, the problem is trivial to solve by replicating the entire repository in cache.

Assume the size of the cache is larger than the largest clip. When a request references a clip j that is not cache resident, the cache manager must bring clip j into cache. The free cache space may be smaller than the size of incoming clip j, requiring the cache manager to swap out other clips to free sufficient space to accommodate clip j.

When an unpopular clip is referenced, a device may stream the clip without storing it in its cache. This prevents the cache manager from swapping out the popular clips from its cache when the unpopular clip is larger than its free space. In this study, we assume the cache manager materializes every referenced clip in its cache for two reasons. First, a device may download a clip faster than its display rate because available network bandwidth is abundant. For example, this may happen when using the Wi-Fi base station at home. Second, a device may anticipate that it will move out of the radio range of a base station. To prevent hiccups, the device may download the clip at a faster rate so that it is cache resident prior to becoming disconnected from the base station. A future research direction is to consider scenarios where the cache manager does not materialize an unpopular clip.

Parameter	Definition
C	Number of clips in the repository
f_i	Frequency of access to clip i
S_i	Size of clip i
S_{DB}	Size of the database, $S_{DB} = \sum_{i=1}^{C} S_i$
S_T	A device's cache size
$B_{Display,i}$	Bandwidth required to display clip i

Table 1. Parameters and their definitions

In this study, we compare and contrast off-line and on-line cache management strategies. Sections 3.1 and 3.2 describe these strategies in turn. Subsequently, Section 3.3 provides a comparison of these strategies with one another.

3 Current caching techniques

This section describes Simple as an off-line technique, and LRU-K and GreedyDual as two on-line strategies. Subsequently, we compare these alternatives with one another, quantifying their tradeoffs.

3.1 An off-line technique: Simple

Simple [11] assumes advanced knowledge of the size and frequency of access to each clip in the database. Given the reference string, Simple computes the frequency of access for clip j (f_j) by counting the number of references to clip j and dividing this value with the total number of requests in the reference string. Next, Simple sorts clips based on their frequency of access to each byte, called byte-freq and defined as $\frac{f_j}{S_j}$. It assigns those clips with the highest byte-freq value to the cache of a device. Experimental results of [11] show the superiority of this technique to one that simply assigns the most frequently accessed clips to each device.

```
GreedyDual(Clip $x$)
 **if**(clip $x$ is cache resident) $Q_x = L + \frac{Cost(x)}{Size(x)}$;
 **else**
 {
   **while**(free cache space $<$ Size($x$))
     {
      $L$ = min($Q_j$);
      Evict $j$ from local cache;
     }
   Retrieve and store clip $x$;
   $Q_x = L + \frac{Cost(x)}{Size(x)}$;
 }
```

Figure 1. GreedyDual pseudo-code

3.2 On-line techniques: LRU-K and GreedyDual

LRU-K [14] maintains the time stamp of last K references to a clip and uses this information to statistically estimate the interarrival times of references to a clip. When choosing a victim, it selects the least recently referenced clip across its last K requests.

GreedyDual [18] is a spectrum of algorithms. One algorithm considers a cache of pages with the same size and different costs to fetch from secondary storage. The algorithm assigns a priority value, Q, with each cached page. When the cache manager materializes a page in the cache, Q is set to the cost of bringing the page into the cache. When the cache is exhausted and the cache manager must choose a victim, it chooses the page with the lowest Q (Q_{min}) value as victim. Next, for each cached page, the cache manager reduces its Q value by Q_{min}. If a page is referenced and observes a cache hit, the cache manager restores its Q value to its original cost value.

With our repository of clips, the priority of a clip is defined as $\frac{cost}{size}$ where size is the size of the clip in bytes. The definition of cost depends on the objective of the cache manager. By setting cost to 1, the cache manager maximizes cache hit rate. One may set cost to the time required to fetch the clip. This would minimize the average latency [3].

A naive implementation of GreedyDual would require the cache manager to perform k subtractions when choosing a victim where k is the number of cache resident clips. An efficient implementation is presented in [3] and shown in Figure 1. The key idea is to maintain an "inflation" parameter L. All values of Q are offset by the value of this parameter.

3.3 A comparison of on-line and off-line caching techniques

We use a simulation model to investigate the behavior of alternative caching strategies. It consists of a server and a client. The server contains the entire database repository and the client is the FMC device configured with a cache that is a fraction of the repository size.

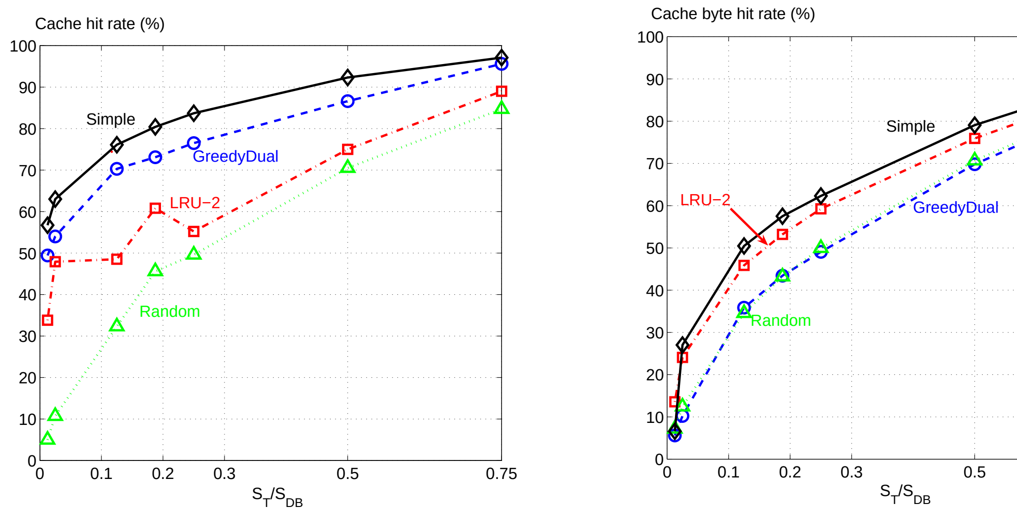
The repository consists of 576 clips. Half are audio clips and the other half are video clips with display bandwidth requirement ($B_{Display}$) of 300 Kbps and 4 Mbps, respectively. The database consists of 3 different clip sizes for each media type. With video, clips have a display time of 2 hours, 60 minutes, and 30 minutes. The size of these clips are 3.5 Gigabytes (GB), 1.8 GB, and 0.9 GB, respectively. With audio, clip display times are 4 minutes (8.8 MB), 2 minutes (4.4 MB), and 1 minute (2.2 MB). We number clips from 1 to 576. This numbering is important because we assume a Zipfian distribution with a mean[4] of 0.27 to generate requests for different clips. Odd numbered clips are video and even numbered clips are audio. Clips are assigned in descending size order in a round-robin manner. Thus, the pattern of clip sizes is 3.5 GB, 8.8 MB, 1.8 GB, 4.4 MB, 0.9 GB, and 2.2 MB. This pattern repeats itself until all 576 clips that constitute the repository are constructed.

We assume the bandwidth between the server and the client exceeds maximum $B_{Display}$ (4 Mbps), enabling the server to stream a clip to the client. A client issues 10,000 requests for clips one after another. A request references a clip using a random number generator conditioned using the Zipfian distribution. We assume the client displays the referenced clip and issues another request immediately. We vary the size of the client's cache (S_T) and report on the client's observed cache hit rate.

Figure 2 shows the cache hit rate and byte hit rate with the alternative techniques as a function of $\frac{S_T}{S_{DB}}$. A larger $\frac{S_T}{S_{DB}}$ (x-axis) value means a larger client cache size, resulting in a higher cache hit rate. As a comparison yard stick, we have included a technique that chooses victims randomly. This technique is called Random. Note that the cache hit rate of this technique also rises with a larger client cache.

With a small client cache size ($\frac{S_T}{S_{DB}}$=0.0125), Simple provides the highest cache hit rate. This is because it packs the clips with the highest byte hit ratio into the client's cache. When the client references an unpopular clip j, Simple swaps out those clips with the smallest byte-hit ratio, enabling clip j to become cache resident. This clip is swapped out immediately by the next clip that is not disk resident. We examined a variant of Simple that does not cache those

[4] This distribution is shown to resemble the sale of movie tickets in the United States [6]. A Zipfian distribution is used extensively to study the behavior of cache servers [16].



2.a) Cache hit rate (%)

2.b) Byte hit rate (%)

Figure 2. A comparison of Simple, LRU-2, GreedyDual and Random.

referenced clips whose byte hit ratio is smaller than the clips occupying client's cache. This variant assumes the referenced unpopular clips are streamed from the server without becoming cache resident. This variant performs either identical or slightly better than the one shown in Figure 2.a.

When considering cache hit rate, both Simple and GreedyDual outperform LRU-2 because they consider size of the clips when deciding what clip to maintain in the cache. This is consistent with the studies investigating caching techniques for web proxy caches [3].

Note that LRU-2 provides competitive byte-hit rates, see Figure 2.b. Except for $\frac{S_T}{S_{DB}}$=0.0125, Simple provides a higher byte-hit rate than LRU-2.

GreedyDual is not a replacement for LRU-K for equi-sized pages or objects. Figure 3 shows LRU-2 provides a higher cache hit rate than GreedyDual for a repository of equi-sized clips. GreedyDual provides a lower hit rate because it ignores the last reference time to a clip as an estimate of its future reference time. To illustrate this, consider three equi-sized clips (o_1, o_2, and o_3) where the size of each clip is 10 MB. Assume the cache is 25 MB in size. Consider a sequence of references by the client where o_1 is referenced every other request. Two consecutive references for o_2 and o_3 are three requests apart, resulting in the following reference string: $\{o_1, o_2, o_1, o_3, o_1, o_2, o_1, o_3, o_1, ...\}$. The first 3 references cause o_1 and o_2 to become cache resident. The subsequent reference to o_3 forces both LRU-2 and Greedy-Dual to choose a victim. LRU-2 will choose o_2 because it is least recently referenced. GreedyDual will have a priority of $\frac{1}{10\ MB}$ for each of o_1 and o_2 and must choose one randomly. Lets give GreedyDual the benefit of choosing o_2

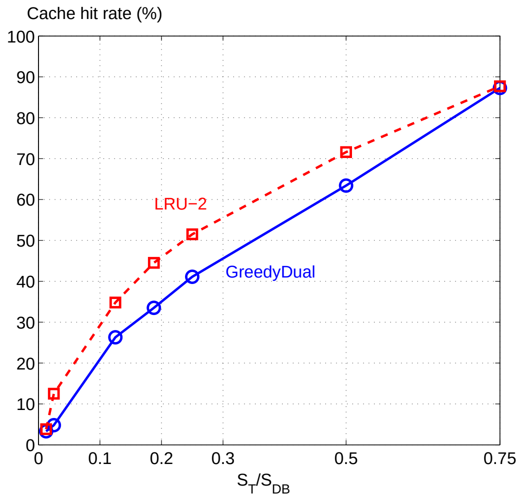


Figure 3. LRU-2 provides a higher cache hit rate than GreedyDual for a repository of equi-sized clips.

as the victim. Once o_3 becomes cache resident, its priority is set to $\frac{2}{10\ MB}$. The next reference for o_1 will observe a cache hit with both LRU-2 and GreedyDual. GreedyDual sets the priority of o_1 to $\frac{2}{10\ MB}$. The next reference for o_2 is a cache miss with both LRU-2 and GreedyDual. LRU-2 will choose o_3 as the victim. GreedyDual must once again choose between o_3 and o_1 randomly because their priorities are identical, $Q(o_1)$=$Q(o_3)$=$\frac{2}{10\ MB}$. This random decision is imposed on GreedyDual every time either o_2 or o_3 is referenced. Every time GreedyDual chooses o_1, its hit rate will be lowered for subsequent o_1 references. LRU-2 will not suffer from the same dilemma because it considers the last 2 references for each clip and will maintain o_1 cache resident.

4 Enhanced greedy caching techniques

In this section, we improve upon the current state of the art techniques to make them more suitable for managing a cache of clips. These improvements are as follows. We introduce a variant of Simple, called DYNSimple, as an online technique. Next, we extend GreedyDual to consider time in order to support a repository of equi-sized clips effectively. The resulting technique is called Interval-Based GreedyDual (IGD). Finally, we present a modified LRU-K, called LRU-SK, which considers clip sizes when choosing victims. In Section 4.4, we compare these techniques with one another.

4.1 Dynamic Simple, DYNSimple

One may transform the Simple technique of Section 3.1 from an off-line technique to an on-line one by dynamically estimating the frequency of access to each clip. The pseudocode for DYNSimple is shown in Figure 4. There are several ways to estimate a clip's frequency of access. Below, we describe one approach.

To estimate the frequency of access to a clip j, a device maintains the time stamp of its last K references: $\{T_{K-1}$, T_{K-2}, ..., $T_0\}$ where $T_{K-1} < T_{K-2}$ and time is increasing monotonically. At time instance t, the arrival rate of requests for clip j is defined as $\lambda_j = \frac{K}{t - T_{K-1}}$. The estimated frequency of access to clip j is the ratio of its arrival rate to the total arrival rate of all C clips, $E(f_j) = \frac{\lambda_j}{\sum_{k=0}^{k=C-1} \lambda_k}$.

One may define the quality of estimated frequencies of access using the following function: $\sqrt{\frac{\sum_{i=1}^{C}(E(f_i)-f_i)^2}{C}}$. $E(f_i)$ is the estimated frequency of access while f_i is the accurate frequency of access obtained from the distribution used to generate requests. A higher value of K improves the quality of estimated f_i values. For a repository of C=576 clip, the quality of estimates improves by a factor of 10 when we increase K from 2 to 60 (quality improves from 0.006 to 0.0006). At the same time, we observed minimal improvements in the cache hit rates with increased values of K, see Section 4.4. We believe $K = 2$ is sufficient in most cases.

```
DYNSimple(Clip $x$)
 if (clip $x$ is cache resident) return clip $x$;
 else
 {
   victims = empty set;
   candidates = set of cached clips;
   sort candidates in ascending order of $\frac{f_j}{size(j)}$;
   while (free cache space
            + size of objects in victims set $< size(x)$)
     {
      victims = victims $\cup$ first clip in candidates list;
      remove first clip in candidates list;
     }
   sort victims in descending order of size;
   while (free cache space $< size(x)$)
     {
      Remove the first clip in victims set;
     }
   Retrieve and store clip $x$;
 }
```

Figure 4. Dynamic Simple (DYNSimple) pseudo-code

Note that Dynamic Simple maintains K time stamps for those clips that are not in its cache. Assuming each time stamp is a 4 byte integer, this technique incurs an overhead of 4 megabytes to maintain $K = 2$ time stamps for one million clips. This might be a reasonable overhead to incur given a cache that is potentially tens of gigabytes in size. To reduce this storage overhead, Dynamic Simple may delete the history of those obsolete clips by employing a rule such as the 5 minute rule [12, 14] extended to the wireless environment. Design of such a rule is a future research direction.

4.2 Interval based GreedyDual (IGD)

Section 3.3 showed GreedyDual does not provide effective support for equi-sized clips because it does not consider how recently a clip has been referenced. Interval based GreedyDual addresses this limitation by maintaining the last K references for each clip j (similar to DYNSimple). At a time instance t, I_K is the interval of time from t to T_{K-1}, $I_K = t - T_{K-1}$. We change the cost function of GreedyDual to $\frac{Ref(x) \times Cost(x)}{I_K \times Size(x)}$ where $Ref(x)$ is the number of references to clip x. While time is monotonically increasing, number of references pertains to those objects in the cache. IGD forgets $Ref(x)$ when clip x is swapped out of the cache. When clip x is referenced and brought into cache, its $Ref(x)$ is set to zero.

The use of $Ref(x)$ was originally proposed in [4]. This study proposed GreedyDual-Freq which changes the cost function of GreedyDual to $\frac{Ref(x) \times cost(x)}{size(x)}$. We use $\frac{Ref(x)}{I_K}$ (instead of $Ref(x)$) in order to facilitate aging so that IGD "forgets" past references when access patterns to clips evolve and change over time. This ratio is not the arrival rate of requests for clip x because the number of requests since a clip becomes cache resident is independent of its K^{th} last reference. To illustrate, $Ref(x) = 100$ means the user of the device referenced this clip 100 times since it was brought into cache. A value of 5 for I_K means 5 time units have elapsed since the user last referenced clip x. Thus, if a popular clip suddenly becomes unpopular and does not receive cache hits, I_K starts to increase, reducing its priority and causing IGD to swap this clip out. When x is swapped out, its number of references is forgotten and reset to zero, similar to [4].

Results of Section 4.4 show IGD is more robust to evolving access patterns than GreedyDual-Freq. Moreover, it provides a higher cache hit rate than GreedyDual.

4.3 LRU-SK

By choosing the least recently used clip, LRU-K swaps out the clip with minimum $\frac{1}{I_K}$ value first. I_K is the interval of time from current time to the last K^{th} reference (same as IGD, see Section 4.2). To consider clip sizes, we propose LRU-SK which swaps out clip x with minimum $\frac{1}{I_K \times size(x)}$ value. Results of Section 4.4 show this change enables LRU-SK to provide cache hit rates higher than GreedyDual.

4.4 A comparison

Figure 5 shows the cache hit rate of the alternative techniques with two different repositories consisting of 576 equi and variable sized clips, respectively. With both figures, x-axis is the ratio of cache size and the database size while the y-axis is the cache hit rate. Figure 5.a shows the revised version of GreedyDual, IGD, provides competitive cache hit rates for a repository of equi-sized objects. Its hit rate is significantly higher than the original GreedyDual and comparable to DYNSimple. When compared with GreedyDual, IGD is an improvement because it considers both the number of references to cache resident clips and their last K^{th} references when selecting victims.

Figure 5.a shows DYNSimple and IGD provide a higher cache hit rate than LRU-2 for equi-sized clips. This is important for the following reason. One may design a technique that a) partitions the cache and each variable sized clip into equi-sized blocks, and b) manages blocks using LRU-2. Such a technique will not provide a higher cache hit rate than either DYNSimple or IGD. Moreover, it wastes space when the chosen block size is larger than an object size, resulting in a lower cache hit rate.

Figure 5.b shows the cache hit rate for a repository of 576 variable sized clips, see Section 3.3. Obtained results show LRU-SK is providing cache hit rates comparable with DYNSimple and GreedyDual. DYNSimple outperforms LRU-SK because DYNSimple employs K=32 references to estimate frequency of access to each clip while K is 2 with LRU-SK. If one employs K=2 with both LRU-SK and DYNSimple then their cache hit rates becomes almost identical. This is because the way they use clip size and reference time to its last 2 requests results in the same ranking of victim clips[5]. With values of $K > 2$, DYNSimple provides a higher cache hit rate than LRU-K.

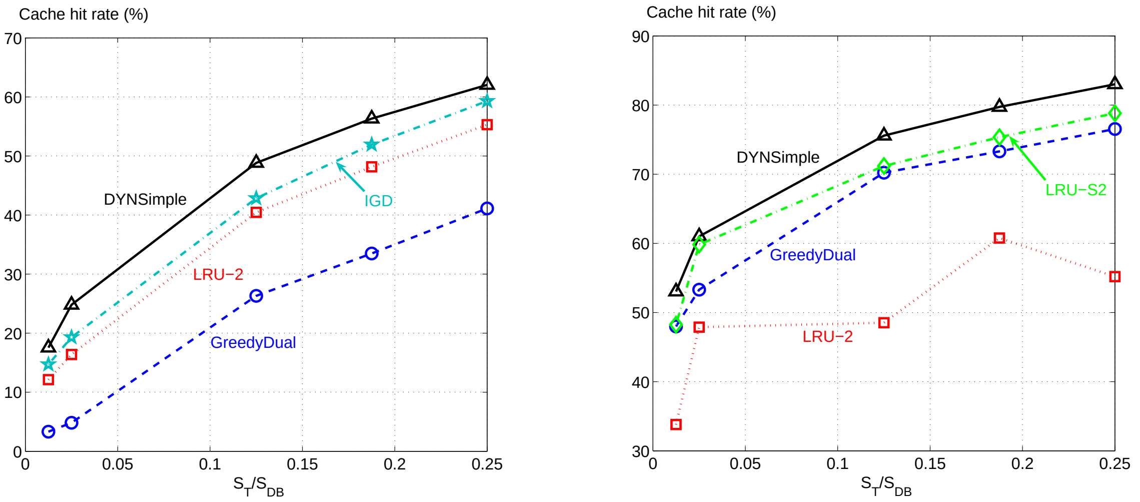
4.4.1 Evolving access patterns

In a final experiment, we analyzed adaptability of the alternative techniques to the changing pattern of access to the 576 variable sized clips. In this experiment, we shifted the identity of objects generated using the Zipfian distribution. Assuming object j is the most popular one with the original distribution, a shift-id of 100, $g = 100$, causes object $(j + 100)\%C$ to become most popular. In essence, we shift the original distribution with the value of g. When $g = 0$, the distribution generates requests identical to the original distribution.

Note that for a g value, we have a fixed distribution of access to clips. We use this distribution to compute the accurate frequency of access to each clip j, f_j. Using this information, our experiment is as follows. A device invokes a replacement policy with 10,000 requests for a given g value. It then employs the accurate frequency of access for each clip j to compute the frequency of access to the cache. This is the theoretical cache hit rate. Assuming k objects are cache resident, this metric is defined as $\sum_{i=1}^{k} f_k$.

Figure 6.a shows the theoretical cache hit rate with the alternative techniques for six different g values: 0, 100, 200, 300, 400, and 500. As a comparison yard-stick, we show Simple which employs the accurate frequency of access to manage the content of the cache. Figure 6.b focuses on two intervals, g=200 and 300, showing the cache hit rate every 100 requests for the 20,000 issued requests. All techniques observe a sharp drop at request id 20,000 because the value of g changes from 200 to 300, modifying the identity of the popular clips. Simple is quick to swap unpopular objects

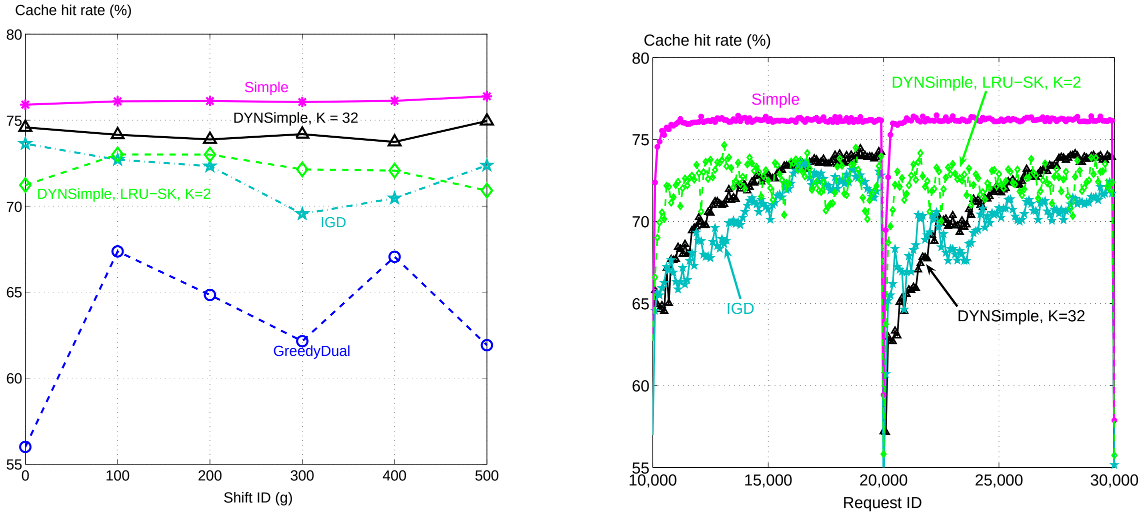
[5] All random number generators for the simulator are seeded, producing a deterministic sequence of requests for all technique.



5.a) Equi-sized clips

5.b) Variable-sized clips

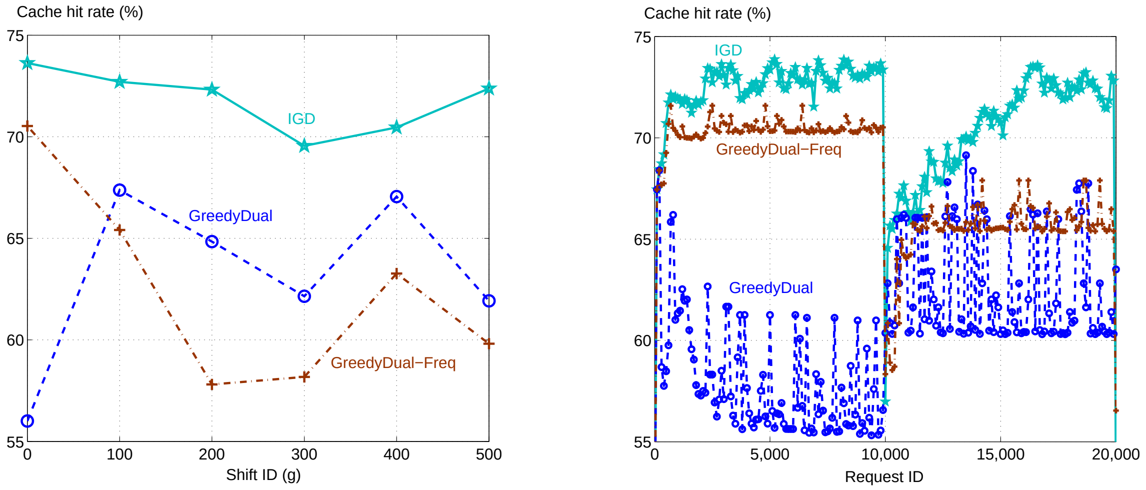
Figure 5. A comparison of DYNSimple, IGD, and LRU-SK.



6.a) After 10,000 requests

6.b) Every 100 requests

Figure 6. Cache hit rate with changing access patterns, $\frac{S_T}{S_{DB}}$=0.125.



7.a) After 10,000 requests

7.b) Every 100 requests

Figure 7. Comparison of IGD with GreedyDual-Freq and GreedyDual, $\frac{S_T}{S_{DB}}$=0.125.

from the cache in favor of the popular clips using their accurate frequency of access, providing the best cache hit rate after two hundred requests. DYNSimple and LRU-SK are sensitive to the chosen value of K. When $K = 2$, DYNSimple and LRU-SK adapt after a few hundred (200 to 400) requests. With a higher value of K, say 32, LRU-SK's hit rate goes down per discussions of Figure 5 (LRU-SK with $K = 32$ is not shown in Figure 6.b). DYNSimple requires a larger number of requests to forget the past and swap out clips that were popular previously. IGD requires a large number of requests to stabilize because its aging function utilizes number of references for clips.

Figure 7 shows a comparison of IGD with GreedyDual [3] and GreedyDual-Freq [4], see Section 4.2. These results show GreedyDual-Freq does not adapt to evolving access patterns as quickly as IGD. With different g values, see Figure 7.a, IGD provides a higher average cache hit rate when compared with GreedyDual-Freq. Indeed, GreedyDual-Freq is worse than the original GreedyDual when g is greater than zero. This is because the number of references to a clip is monotonically increasing with GreedyDual-Freq as long as it is cache resident. However, IGD causes the number of references to age by dividing it with the interval of time since last reference.

When the access pattern is fixed, GreedyDual-Freq provides a cache hit rate comparable to IGD. This is shown in the first half of Figure 7.b. It is interesting to note that GreedyDual results in different cache hit rates as a function of every 100 requests. We speculate this is because GreedyDual does not have the concept of least recently used clip and at times chooses the wrong clip to swap out, see the discussions of Section 3.3.

We manipulated the mean of the Zipfian distribution used to generate clip-ids referenced by different requests to analyze the impact of a skewed pattern of access. In general, the cache hit rate with the alternative techniques becomes almost identical with a more skewed access pattern. With a more uniform distribution of access, DYNSimple outperforms the other techniques by a wider margin.

5 Conclusions and future research directions

In this study, cache hit rate quantifies percentage of requests serviced successfully when a mobile device is not in the radio range of a base station. Our contributions are as follows. First, we identify limitations of current caching techniques by comparing them with one another when the objective is to maximize cache hit rate. Our second contribution is to introduce three novel techniques: Dynamic Simple (DYNSimple), Interval based GreedyDual (IGD), and a variant of LRU that considers size when choosing victims (LRU-SK). These variants address the limitations of their original counter-parts in two ways. First, they support repositories of equi-sized and variable-sized clips. Second, they adapt to changes in access patterns and do not suffer from cache pollution, i.e., the tendency of previously popular clips lingering in the cache. When compared with one another, DYNSimple provides a higher cache hit rate and adapts faster when it computes frequency of access to clips based on their past two references.

One may argue that increasing cache hit rate by several percentage points is negligible. Such a conclusion is ill-guided because several studies have shown that cache hit rate grows as a log function of cache size [1, 3, 2, 13]. Thus, a better algorithm that increases cache hit rate by only several percentage points would be equivalent to several fold increase in cache size [13].

We are extending this study in several directions. First, we plan to develop efficient implementations of DYNSimple, IGD, and LRU-SK. This may require tree-based data structures to minimize the complexity of identifying a victim clip. Second, some applications may not tolerate the storage overhead of maintaining last reference times for all clips with DYNSimple. We propose to develop a rule similar to the 5 minute rule that considers the economics of network bandwidth and local storage, providing a framework that decides how long to keep the meta-data for the past references.

Finally, none of the presented techniques are cooperative. Cooperative caching techniques enable mobile devices to coordinate their state in order to optimize a global criterion such as number of references serviced without accessing the base station. Recall from Section 1 that a FMC phone includes a Wi-Fi networking cards. Multiple devices in the same radio range may form an ad hoc network and exchange clips with one another. They may employ a cooperative caching technique to minimize the number of references to the base station. Design of an effective cooperative caching technique and its comparison with a greedy technique remains a future research direction.

References

[1] V. Almeida, A. Bestavros, M. Crovella, and A. Oliveira. Characterizing Reference Locality in the WWW. In *Proceedings of International Conference on Parallel and Distributed Information Systems*, December 1996.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proceedings of Infocom*, pages 126–134, 1999.

[3] P. Cao and S. Irani. Cost-Aware WWW Proxy Caching Algorithms. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA, 1997.

[4] L. Cherkasova and G. Ciardo. Role of Aging, Frequency, and Size in Web Cache Replacement Policies. In *Proceedings of the High-Performance Computing and Networking*, December 2001.

[5] S. M. Cherry. WiMax and Wi-Fi: Separate and Unequal. *IEEE Spectrum*, 41(3):16–16, March 2004.

[6] A. Dan, D. Dias, R. Mukherjee, D. Sitaram, and R. Tewari. Buffering and Caching in Large-Scale Video Servers. In *Proc. of COMPCON*, 1995.

[7] P. J. Denning. The Working Set Model for Program Behavior. *Communications of the ACM, Vol. 11, No. 5*, pages 323–333, 1968.

[8] The Economist. A Survey of Telecoms Convergence, October 14, 2006.

[9] M. J. Feeley, W. E. Morgan, F. H. Pighin, A. R. Karlin, H. M. Levy, and C. A. Thekkath. Implementing Global Memory Management in a Workstation Cluster. In *Proceedings of the 15th ACM Symp. on Operating Systems Principles*, December 1995.

[10] S. Ghandeharizadeh, A. Dashti, and C. Shahabi. Pipelining Mechanism to Minimize the Latency Time in Hierarchical Multimedia Storage Managers. *Computer Communications*, 18(3):38–45, March 1995.

[11] S. Ghandeharizadeh, T. Helmi, T. Jung, S. Kapadia, and S. Shayandeh. An Evaluation of Two Policies for Simple Placement of Continuous Media in Multi-hop Wireless Networks. In *Twelfth International Conference on Distributed Multimedia Systems (DMS)*, August 2006.

[12] J. Gray and F. Putzolu. The 5 Minute Rule for Trading Memory for Disc Accesses and 10 Byte rule for Trading Memory for CPU Time. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, 1987.

[13] S. Jin and A. Bestavros. Popularity-Aware GreedyDual-Size Web Proxy Caching Algorithms. In *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS)*, pages 254–261, April 2000.

[14] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm for Database Disk Buffering. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 1993.

[15] S. Podlipnig and L. Boszormenyi. A Survey of Web Cache Replacement Strategies. *ACM Computing Surveys*, 35(4):374–398, December 2003.

[16] A. Wolman, M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy. On the Scale and Performance of Cooperative Web Proxy Caching. *SIGOPS Oper. Syst. Rev.*, 33(5):16–31, April 1999.

[17] K. Wong. Web Cache Replacement Policies: A Pragmatic Approach. *IEEE Network*, pages 28–34, January/February 2006.

[18] N. E. Young. On-line Caching as Cache Size Varies. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1991.